IMOCA : une architecture à base de modes de fonctionnement pour une application de contrôle dans un environnement incertain

Goulven Guillou, Jean-Philippe Babau

IMOCA : une architecture à base de modes de fonctionnement pour une application de contrôle dans un environnement incertain

G. Guillou
Lab-STICC/UMR 6285, UBO, UEB
20 Avenue Le Gorgeu
29200 Brest, France
goulven.guillou@univ-brest.fr

J.P. Babau
Lab-STICC/UMR 6285, UBO, UEB
20 Avenue Le Gorgeu
29200 Brest, France
jean-philippe.babau@univ-brest.fr

Résumé

Les systèmes de contrôle de processus par régulation font intervenir plusieurs modes de fonctionnement (lois de commandes) en fonction du contexte. Dans un environnement naturel fortement perturbé, il est alors nécessaire de répondre aux trois questions suivantes vis-à-vis des modes de fonctionnement :

1. Quels sont les modes de fonctionnement pertinents ?
2. Comment enchaîner plusieurs modes ?
3. Comment paramétrer les modes ?

Pour répondre à ces questions, l'approche proposée s'appuie sur une architecture logicielle et une méthodologie épaulées par des outils de simulation et de mise au point incrémentale par essais/corrections. Le choix des modes de fonctionnement, la manière de les enchaîner et de les paramétrer reposent sur une expertise. Les différents modes (lois de commande) sont associés à des contextes c'est-à-dire à des états de l'environnement. L'enchaînement des lois de commande s'effectue par commutation et est géré par un automate. Enfin les différents paramètres des modes dépendent du contexte et sont tabulés. L'architecture présentée est indépendante des technologies des capteurs et des actionneurs (interaction avec l'environnement). La partie contrôle est composée de trois contrôleurs dits `réactif`, `expert` et `adaptatif` fonctionnant de manière parallèle et asynchrone. Le contrôleur réactif est chargé de l'application temps réel d'une loi de commande choisie par le contrôleur expert et paramétrée par le contrôleur adaptatif.

L'approche est validée sur le pilotage automatique d'un voilier en s'appuyant sur un environnement virtuel pour la simulation.

Mots Clef

Architecture, contrôle, adaptation, contexte, modes

1. INTRODUCTION

Les systèmes de contrôle de processus sont de plus en plus utilisés pour assister l'activité humaine, y compris dans des environnements incertains. Dans ce contexte, disposer d'outils d'aide au développement et de simulation devient crucial. Un système de contrôle utilise des capteurs pour obtenir des informations sur le processus à contrôler ainsi que sur son environnement physique (voir figure 1). D'autre part, des actionneurs permettent d'appliquer les commandes déterminées par le système de contrôle. De tels systèmes sont souvent spécifiques à chaque domaine, voire à chaque processus à cause des contraintes liées à l'environnement ainsi que celles liées aux matériels eux-mêmes. Ils sont donc souvent développés au cas par cas, ce qui pose des problèmes de mise au point et au final de déploiement. Les méthodes

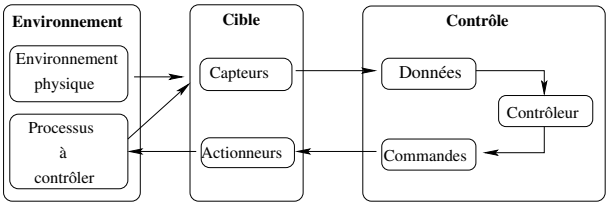


Figure 1: Système de contrôle

orientées objets et, plus récemment, l'ingénierie dirigée par les modèles (IDM [2, 20]) offrent un cadre pour la maîtrise du développement de ce type d'applications. En particulier, l'architecture dirigée par les modèles (MDA pour Model Driven Architecture [17]) se concentre plus particulièrement sur la séparation des problèmes en proposant trois modèles appelés PIM [1], PM [2] et PSM [3]. Par la suite on appellera environnement le processus ainsi que son environnement physique.

Dans ce contexte, l'architecture IMOCA (pour archItecture for MOde Control Adaptation) se focalise sur les systèmes de contrôle de processus dans un environnement naturel perturbé où l'intégrité physique du processus peut être remise en cause. Les avions, les engins spatiaux, les véhicules terrestres ou lunaires, les bateaux ou encore les drones sont des exemples typiques de cette catégorie de processus.

1. Platform Independent Model
2. Platform Model
3. Platform Specific Model

Dans ces systèmes, les lois de contrôle font intervenir plusieurs modes de fonctionnement ou lois de commande soit parce que plusieurs actionneurs sont présents (pour mouvoir un bras articulé et entraîner des roues par exemple) soit parce qu'il est important de pouvoir agir de différentes manières sur un même actionneur (ne serait-ce que pour passer d'un état de marche à un état d'arrêt par exemple). Dans la pratique peu d'entre eux s'avèrent utilisables, ce qui nécessite de répondre aux trois questions suivantes :

1. Quels sont les modes de fonctionnement pertinents ?
2. Comment enchaîner plusieurs modes ?
3. Comment paramétrer les modes ?

Ce document présente comment intégrer, dans une approche MDA avec l'architecture IMOCA, les spécificités d'un système de contrôle tout en répondant à ces trois questions. L'architecture IMOCA, parce qu'il est important de séparer l'application de contrôle du processus lui-même, est basée sur trois couches appelées cible, interprétation et contrôle. Le contrôleur est lui-même composé de trois contrôleurs dits `réactif`, `expert` et `adaptatif` fonctionnant de manière parallèle et asynchrone. Le contrôleur réactif est chargé de l'application temps réel d'une loi de commande choisie par le contrôleur expert et paramètrée par le contrôleur adaptatif.

La suite de ce document est organisée comme suit : après une présentation de l'état de l'art, l'architecture IMOCA est décrite en détail puis une méthodologie destinée à aider le concepteur du système de contrôle à choisir, combiner et paramétrer les modes est fournie. Enfin, en guise de validation, une solution pour le cas du pilotage automatique des voiliers est développée conformément à la méthodologie et à l'architecture présentées.

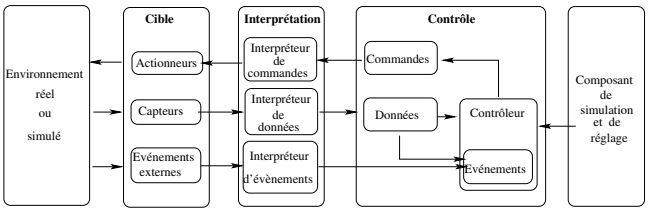
2. ETAT DE L'ART

IMOCA est une architecture [5] pour décomposer un système en sous-systèmes en offrant des outils conceptuels pour la maîtrise (mise au point) de chaque partie. De ce fait, IMOCA intègre divers paradigmes existants, vus comme des briques de base conceptuelles pour l'architecture. D'abord, une séparation claire est proposée entre la plateforme (la cible qui correspond à la plateforme réelle) et l'application (le contrôle qui s'appuie sur une plateforme abstraite). On reprend ici le paradigme MDA [17, 7] et plus précisément le style architectural SAIA [8] où les entrées/sorties de haut-niveau (les données du contrôle) sont séparées des données de bas niveau (le monde vu au travers des capteurs/ actionneurs) [16]. Cette dernière vue correspond à une plateforme abstraite explicite [14]. Au niveau du contrôle, IMOCA introduit plusieurs niveaux à travers divers contrôleurs [13] pour permettre une adaptation au contexte. La politique d'adaptation est de type Model Reference Adaptive Control [4] en s'appuyant sur une observation de l'environnement et une expertise du contrôle pour assurer une robustesse dans le comportement adaptatif. Le contrôle s'appuie lui sur des modes de fonctionnement paramétrables. Les modes de fonctionnement sont classiques dans les applications de contrôle [18, 9]. Ils permettent *via* des automates (le contrôleur expert) de définir une politique d'adaptation en fonction d'événements (ici des événements externes ou des changements détectés dans l'environnement). IMOCA introduit une autre capacité d'adaptation *via* l'adaptation des paramètres des politiques de contrôle. Cette adaptation est liée au contexte et est pré-tabulée comme dans de nombreux systèmes embarqués *via* des look-up tables [10], essentiellement pour des raisons de performance et de prédictibilité. Il existe beaucoup de travaux concernant les modes dans les architectures, par exemple dans AADL. Ces travaux apportent soit des outils de formalisation à base d'automates, par exemple des réseaux de Petri temporisés, pour la validation du comportement (absence de deadlock, comportement temporel) des systèmes comme dans [1] ; soit des outils de gestion de la reconfiguration des modes à l'exécution comme dans [3]. Pour sa part, IMOCA s'intéresse à l'aide à la mise en place (choix et mise au point) des modes en se concentrant sur l'aspect applicatif (qualité de contrôle). IMOCA est complémentaire de ces approches. L'originalité d'IMOCA est de proposer un composant de mise au point des tables pour le contrôle [15] par simulation *via* une interface dédiée. Enfin, l'utilisation de modèles [20] permet de se concentrer sur les données du problème pour générer l'application et son environnement de mise au point.

3. ARCHITECTURE IMOCA

3.1 Principes de l'architecture IMOCA

L'architecture IMOCA est basée sur SAIA [7] qui est elle-même une approche MDA pour permettre le développement d'un système de contrôle de processus indépendamment d'une technologie de capteurs et d'actionneurs. Cette indépendance est primordiale pour limiter l'impact sur l'application d'un changement de capteur ou d'actionneur. En effet, dans un environnement fortement perturbé, c'est-à-dire soumis à de rapides et importantes variations (température, pression, accélération ...) les pannes peuvent être courantes. De plus, dans un contexte économique concurrentiel, les optimisations de matériel sont incessantes. L'architecture IMOCA présentée figure 2 est constituée de trois couches dites `Cible`, `Interprétation` et `Contrôle`. La `cible` est un modèle de la plateforme matérielle et le `contrôle` utilise une vue idéale de l'environnement, les `données`, pour prendre des décisions appelées `commandes` qui sont transmises aux `actionneurs`. L'adaptation entre la `cible` et le `contrôle` est réalisée par la couche d'`interprétation` qui se charge de faire le lien entre `capteurs` et `actionneurs` d'un côté, et `données` et `commandes` de l'autre côté. Cette configuration assure l'indépendance de la partie contrôle vis-à-vis de la plateforme matérielle. Des `événements externes` peuvent intervenir (panne de capteur, changement de consigne par exemple). Ils influencent directement le contrôle. Enfin, un composant de



Les cadres représentent des composants, les flèches décrivent des flux d'information

Figure 2: Principes de l'architecture IMOCA

simulation et de réglage est ajouté fournissant les outils nécessaires à la mise au point de l'application de contrôle. Il est apte à prendre la main sur cette dernière afin de permettre à

un expert de tester différentes lois de commande, de valider leurs enchaînements et de règler leurs paramètres.

Les principaux composants de l'architecture IMOCA sont détaillés ci-après.

3.2 Données

Pour pouvoir choisir et paramétrer la bonne loi de commande, l'application de contrôle a besoin de se faire une image de l'environnement appelée également contexte. La construction du contexte se fait en deux étapes. Dans la première, l'environnement est mesuré à l'aide de `capteurs`. Les données issues de ces capteurs sont ensuite transformées en `données` de haut-niveau. Au sein de la `cible`, le capteur se réduit à des spécifications purement techniques masquant les aspects bas-niveau. Un `capteur` se caractérise par (voir figure 3 où un capteur est dénoté `Sensor`) un nom `sensorName`, une valeur courante `value` de type `double`, une fréquence `frequency`. Des méthodes `set_data` et `get_data` de mise à jour et de lecture de la valeur du capteur existent pour chaque classe. Ces méthodes n'apparaissent pas dans le métamodèle car leur signature et comportement par défaut sont générées automatiquement. Le comportement est complété par la suite en fonction de l'expertise métier. Les `capteurs` délivrent des données brutes qu'il va falloir interpréter pour obtenir des `données` de haut-niveau. L'ensemble des `données` est déterminé par expertise du domaine d'application en répondant à la question "Quelles sont les données qui permettent de prendre des décisions par rapport au contrôle ?". Il s'agit de données de haut-niveau qui ne comportent aucun détail sur la manière de les obtenir. Les `données` ou `data` figure 3 sont caractérisées par un nom `dataName`, un `type`, une valeur `value`, une fréquence `frequency` ainsi que des opérations de mise à jour et de lecture. Deux grandes catégories de données sont distinguées, les données continues `ContinuousData` pouvant prendre toute valeur entre deux bornes `minimalBound` et `maximalBound` et les données de type énuméré `EnumeratedData` ne pouvant prendre qu'un nombre fini de valeurs. Certaines données, pour indiquer une modification de l'environnement, peuvent générer un événement.

L'`interpréteur de données` est chargé de faire le lien entre les `capteurs` et les `données` (figure 4). Les données issues des `capteurs` sont filtrées, on peut choisir par exemple de ne prendre qu'une donnée sur trois ou d'effectuer une moyenne sur fenêtre glissante ou encore d'utiliser un filtre de Kalman. Ensuite la valeur obtenue est formatée c'est-à-dire convertie dans une unité du Système International. Cela comporte un décalage d'offset ainsi qu'une mise à l'échelle par multiplication par un coefficient de calibration. Ensuite la valeur est stockée dans un buffer circulaire permettant des traitements d'ordre statistique : moyenne, écart type, pente de la droite de régression... Enfin une phase de combinaison permet de fournir une donnée à la couche de contrôle. Cette phase peut être réduite à la transmission de la valeur filtrée et formatée si la donnée nécessaire au contrôle correspond à la donnée délivrée par le capteur.

A partir des données, le contrôle agit pour produire les commandes. La couche de contrôle est maintenant détaillée.

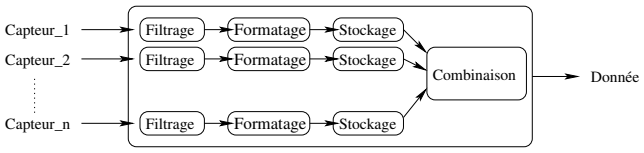


Figure 4: La couche d'interprétation pour une donnée

3.3 Contrôle

3.3.1 Principes

Le contrôle s'effectue à l'aide de lois de commande ou modes de fonctionnement. L'application doit toujours être capable d'appliquer un mode dit courant. D'autre part, il est connu que les lois de commande sont très sensibles aux variations de l'environnement. Il s'avère donc nécessaire soit de changer de mode courant (pour répondre à la question 2 de l'introduction "Comment enchaîner plusieurs modes ?"), soit de modifier ses paramètres (pour répondre à la question 3 de l'introduction "Comment paramétrer les modes ?"). Pour cela, l'application de contrôle est elle-même composée de trois entités fonctionnant en parallèle et de manière asynchrone (figure 5). La première dite `réactiveController` est chargée à haute fréquence (fréquence du capteur le plus rapide, environ $100Hz$) du contrôle temps réel du système, elle utilise des données qui correspondent directement à celles des capteurs (filtre élémentaire sans phase de combinaison dans la couche d'interprétation) pour calculer les commandes à appliquer à partir du mode imposé par l'`expertController`. Ce dernier est contrôlé par un automate (liste de `Transition`) qui change d'état sur un `trigger` (événement) (figure 6). Enfin, l'`adaptativeController` ajuste, à basse fréquence (de l'ordre de $0.1Hz$), les différents paramètres de la loi de commande afin de la rendre plus performante par rapport à l'environnement mesuré.

3.3.2 Contrôleur réactif et modes

Les `Modes` (voir figures 5 et 6) se caractérisent par un nom `modeName`, un ensemble de paramètres `configuration`, un ensemble de consignes `setpoints` et une loi de commande `control`. Le contrôleur réactif effectue une boucle de type perception/action qui consiste à lire un certain nombre de données pour nourrir un mode imposé par le contrôleur expert et délivrer une commande. La loi de commande est une fonction qui dépend à la fois de paramètres, de données directement interprétables et de consignes qui proviennent du contrôleur expert. Par exemple, pour une régulation de type PID de vitesse d'un véhicule, les paramètres sont les coefficients P, I et D (voir figure 5), les données la vitesse du véhicule et la consigne est fournie par le conducteur. Des `data` sont dédiées à la surveillance de données capteurs (temps réel) qui sont sensées rester dans un certain intervalle de valeurs afin d'assurer une utilisation robuste de la loi de commande. En cas de dépassement un évènement est généré. Dans le cas par exemple du pilotage d'un voilier, si la régulation se fait sur l'angle de gîte, il est important de surveiller le cap du bateau pour éviter de gros écarts de route [11]. Cette surveillance doit se faire sur des données temps réel pour obtenir une bonne réactivité car un changement de contexte est par essence lent à repérer. Pour une donnée issue d'un capteur, sortir de l'intervalle spécifié signifie que la loi de commande n'est plus adaptée. Le contrôleur expert

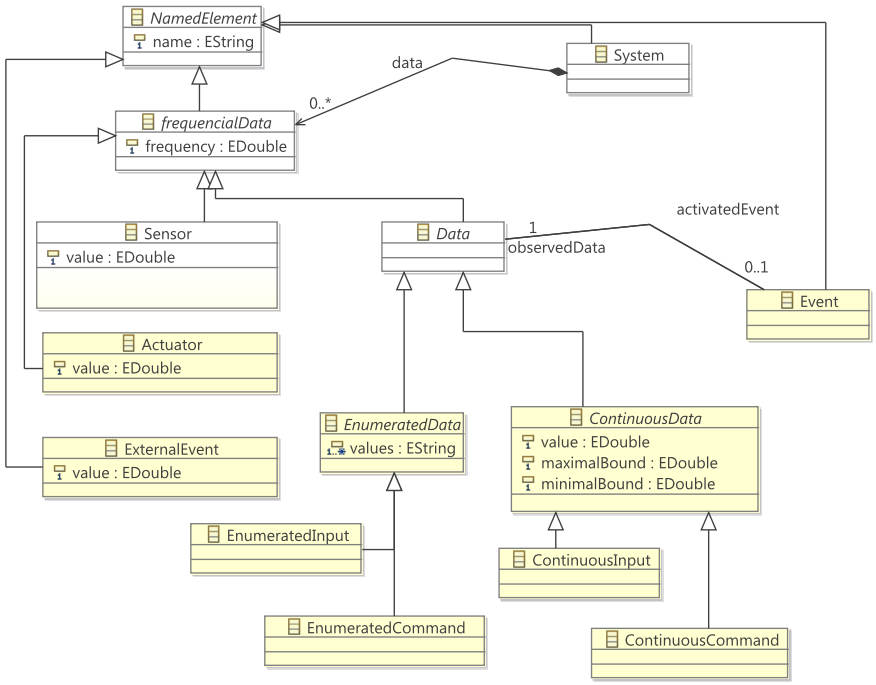


Figure 3: Métamodèle simplifié des données, commandes, capteurs, actionneurs et évènements

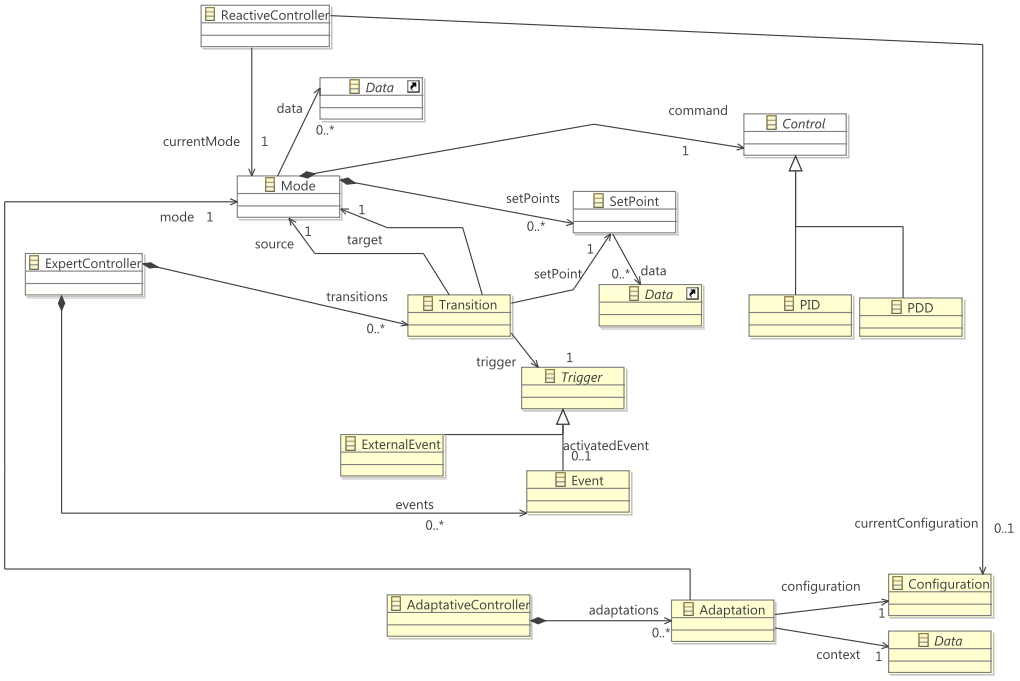


Figure 5: Les différents éléments du contrôleur

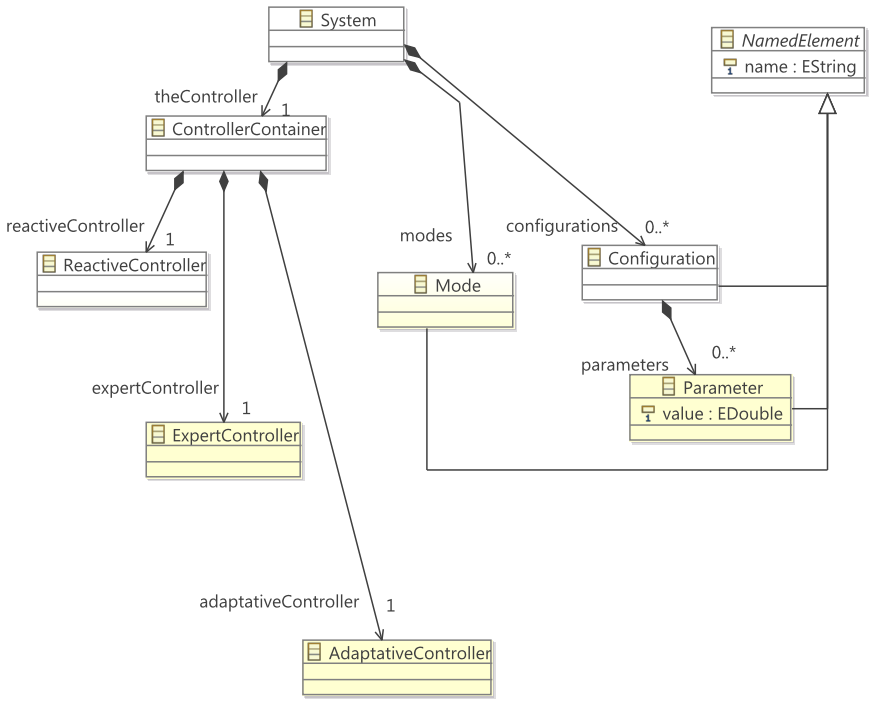


Figure 6: Contrôle et modes

commute alors sur une autre loi. Cette nouvelle loi peut être prévue dans l'enchaînement normal des lois pour l'accomplissement d'une tâche donnée, ou être une loi par défaut dans le cas où le comportement observé n'est pas conforme à l'expertise effectuée. Il est donc important de disposer d'au moins une loi robuste à la plupart des contextes pour pouvoir s'y réfugier en cas de nécessité.

Pour prendre en compte les évolutions de l'environnement, il faut être capable d'adapter les paramètres des modes afin de maintenir un bon niveau de performance dans le nouveau contexte. En effet, plutôt que de changer de mode, il se peut qu'un changement de paramétre soit plus pertinent. Par exemple, dans le cadre de la régulation en cap d'un voilier, l'augmentation de l'amplitude des angles de barre peut suffire pour répondre à l'augmentation de celle des vagues. C'est le contrôleur adaptatif présenté dans la partie suivante qui est chargé de cette adaptation.

3.3.3 Contrôleur adaptatif et table des paramètres

Le contrôleur adaptatif repère les caractéristiques basses fréquences de son environnement pour paramétrer les modes. Il utilise pour cela (figure 5) une table `Adaptation` donnant pour chaque mode et chaque contexte identifié (une `Data`) une `configuration`, c'est-à-dire les valeurs des différents paramètres intervenant dans le mode. La table est construite par expertise à l'aide notamment du composant de simulation et de mise au point. Un exemple de table des paramètres est la table 2 présentée dans la section 5.

Afin de garantir une utilisation cohérente de la table des paramètres il est nécessaire que deux contextes distincts ne puissent être valides simultanément. Enfin si aucun contexte proposé ne correspond au contexte courant il est important de prévoir des paramètres par défaut.

Le contrôleur expert, présenté maintenant, se charge de l'enchaînement des modes qui dépendent à la fois du type d'actionneur commandé, des objectifs à atteindre mais également de l'environnement.

3.3.4 Contrôleur expert et automate

Le contrôleur expert sélectionne la loi de commande et les consignes à appliquer. C'est un automate fini déterministe `Transition` (figure 5) muni d'un état initial où chaque état est associé à une loi de commande. Ses transitions sont étiquetées par des événements et lors d'un changement d'état (`trigger`) les consignes `setPoint` sont évaluées afin de pouvoir effectuer la commutation de modes. Ces consignes sont définies par expertise. Un exemple simple de comportement pour le contrôleur expert est modélisé figure 7 à l'aide de réseaux de Petri interprétés [6]. Bien qu'il n'y ait pas ici de parallélisme sous-jacent, nous utilisons ce formalisme car il est couramment employé pour la modélisation des systèmes à événements discrets [12]. Une place correspond à un état et est associée à un mode. Nous avons ici un comportement simple basé sur deux modes. Le jeton indique le mode courant. Les transitions sont étiquetées par un événement `evt_i` et une fonction `c_i` servant à calculer les consignes. Lorsqu'un événement arrive et qu'une transition validée est étiquetée par cet événement, la fonction `c_i` est exécutée et la transition tirée.

3.3.5 Evènements

Les événements sont soit externes (mise en route du système ou changement de consigne par exemple), soit générés par

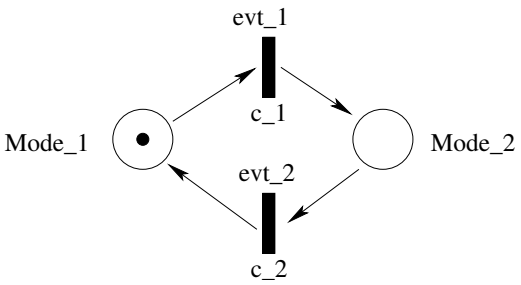


Figure 7: Exemple de comportement simple pour le contrôleur expert

certaines `data`. Ils indiquent un changement de contexte qui nécessite une ré-évaluation du mode courant, qui peut ne plus être approprié. Les événements provoquent l'évaluation de la fonction de transition de l'automate, la récupération des paramètres associés au mode obtenu et l'évaluation des consignes.

3.3.6 Modèles et code

Une fois les modèles décrits, le développeur peut s'appuyer sur un certain nombre de librairies métiers fournies aussi bien pour la couche `interprétation` (conversions usuelles, calibration, offset, outils statistiques, filtres ...), que pour la couche `contrôle` (contextes, automates, tables ...) ou encore pour le composant de simulation et de réglage (widgets, IHM ...). Ainsi, une fois l'architecture spécifiée, le squelette du code est obtenu par génération de code, puis le code final est obtenu par intégration et adaptation des librairies proposées. Pour des raisons de place, cette partie, basée sur des transformations de modèles, n'est pas détaillée dans cet article.

4. MÉTHODOLOGIE

L'utilisation d'IMOCA ne garantit pas en soi que l'application de lois de contrôle réponde aux trois questions posées en introduction :

1. Quels sont les modes de fonctionnement pertinents ?
2. Comment enchaîner plusieurs modes ?
3. Comment paramétrer les modes ?

Le choix des modes de fonctionnement (question 1) possibles est d'autant plus difficile que de nombreuses données sont disponibles et combinables. Cependant, toute loi de commande n'est pas bonne, encore faut-il qu'elle ait un sens (on ne peut décemment contrôler la vitesse d'une voiture en fonction de la température extérieure par exemple), et qu'elle soit réellement efficace, c'est-à-dire, que le contrôle résultant soit de bonne qualité. La mesure de la qualité d'un contrôle est également un problème difficile car de nombreux critères, parfois qualitatifs plutôt que quantitatifs peuvent intervenir : robustesse, sûreté, sécurité, consommation, réponse à un échelon, conformité à une expertise effectuée de l'activité concernée... Le problème de l'enchaînement des lois de commande (question 2) est également très difficile car dépendant de l'environnement, de la tâche à effectuer et de critères parfois difficilement quantifiables comme "avoir une belle trajectoire". Pour répondre à ce problème, la méthodologie proposée repose sur une expertise pour le choix des modes de fonctionnement (question 1), la manière de les enchaîner (question 2) et de les paramètrer (question 3). L'expert est aidé lui-même par des outils de simulation et de mise au point incrémentaux par essais/corrections *via* le composant de simulation et de réglage. La méthodologie comporte 4 étapes :

Etape 1 mise à disposition d'un environnement réel ou simulé, construction de la couche `cible` et du `contrôleur réactif`.

Etape 2 construction de la bibliothèque de lois de commande et de la table des paramètres

Etape 3 caractérisation des données ou contextes en fonction des capteurs

Etape 4 construction de l'automate et définition des évènements

La première étape a pour but de pouvoir jouer avec le processus à contrôler. Le coût des plateformes réelles ainsi que les contraintes et les risques liés à leur utilisation font qu'il est avantageux de simuler le processus ainsi que son environnement pour permettre le rejeu et ainsi tester différentes configurations. L'expert est d'autant mieux épaulé que la simulation est réaliste. Dans ce cadre, la réalité virtuelle par son côté immersif est particulièrement intéressante. Une fois cet environnement disponible la couche `cible` peut être développée ainsi que le contrôleur réactif qui peut être en lien direct avec les capteurs dans un premier temps.

Ensuite la construction d'une interface permettant de choisir et construire différentes lois de commande permet à un expert de sélectionner celles qui semblent les plus pertinentes. Cette interface prend en lieu et place le rôle du `contrôleur expert` et permet de définir les `données` nécessaires aux lois de commande sélectionnées. L'expert teste les lois dans différentes conditions et définit les paramètres qui paraissent les plus adaptés. L'établissement des paramètres conduit à la définition des contextes dans lesquels ils opèrent efficacement et à la construction du `contrôleur adaptatif`.

Dans une troisième étape, les contextes doivent être caractérisés en fonction des capteurs ce qui entraîne la définition de la couche d'interprétation de l'architecture.

On peut ensuite s'intéresser à la construction de l'automate, c'est-à-dire à la construction du `contrôleur expert`, pour lequel, l'intervention d'un expert est primordiale. En effet, l'enchaînement des modes dépend des tâches à effectuer (aller chercher un objet nécessite de se déplacer puis de manipuler un bras et enfin de revenir par exemple), de contraintes mécaniques (s'arrêter avant de pouvoir reculer par exemple et pour s'arrêter il peut être nécessaire de freiner avant) ou encore d'une analyse métier de comment il faut faire. L'élaboration de l'automate amène à définir les différents événements qui vont permettre de changer d'état. Ces événements sont liés d'une part à des changements de contexte et, d'autre part, à la surveillance temps réel de certains capteurs. Il est alors souvent nécessaire de définir de nouvelles `données` et de les relier aux `capteurs`.

5. UNE ÉTUDE DE CAS : LE PILOTAGE AUTOMATIQUE DES VOILIERS

Dans cette section l'architecture IMOCA et la méthodologie proposée précédemment sont mises en oeuvre dans le but de développer un pilote automatique de voilier. C'est un problème crucial que d'affranchir le marin d'une tâche

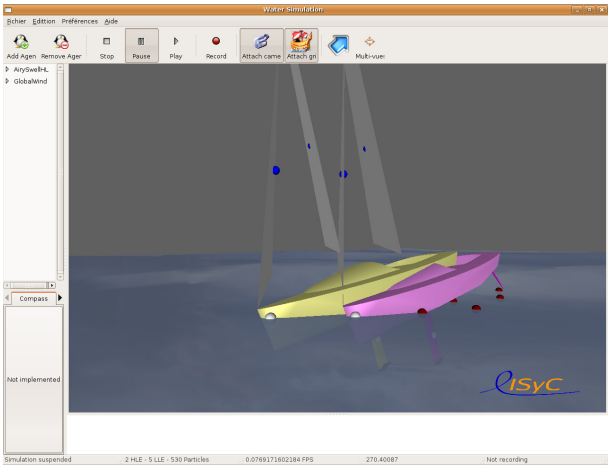


Figure 8: Le bateau (en vert) et son clone (en mauve)

répétitive et fastidieuse tout en veillant à sa sécurité et en préservant une conduite performante le tout dans un environnement, mer, vent, voilier, par essence très perturbé. Les lois de commande choisies sont de type PID. La validation repose sur l'utilisation d'un environnement virtuel offrant une excellente qualité d'expérience et permettant de comparer différents types de pilotage.

5.1 Position du problème

En course à la voile au large et en solitaire le skipper pour pouvoir manœuvrer, régler, manger ou tout simplement dormir utilise un pilote automatique. Les pilotes automatiques commerciaux utilisés sont de simples régulateurs de cap et d'allure qui n'agissent que sur la barre *via* un unique actionneur. Ils sont peu performants et conduisent parfois à des situations critiques entraînant des dégats irréversibles comme des démâtages ou des chavirages. L'amélioration de ces pilotes est une demande forte des coureurs ainsi que des industriels les concevant.

5.2 Construction du pilote

5.2.1 *Etape 1 : mise à disposition d'un environnement réel ou simulé, construction de la couche cible et du barreur réactif*

Pour la mer et le vent le modèle `IPAS` [19] pour Interactive Phenomenological Animation of the Sea implémenté dans le langage `ARéVi` [4] est utilisé. Un modèle de voilier, basé sur la dynamique des systèmes matériels, a été développé ; sa géométrie est définie *via* un fichier VRML qui permet un rendu de qualité. Par défaut, le bateau est muni d'un pilote de type commercial, c'est-à-dire un régulateur de cap. L'environnement offre la possibilité de cloner le bateau pour effectuer des comparaisons de performance sur la loi de contrôle. On peut voir figure 8 le bateau (en vert) dans son environnement physique avec son clone (en mauve).

Le bateau est équipé de tous les capteurs `Sensor` habituels : anémomètre, girouette, speedomètre, GPS, compas, sondeur, angle de barre, plus quelques uns plus spécifiques comme une centrale inertielle ou des jauges de contrainte [5]. Un actionneur `Actuator`, un vérin linéaire agissant sur les safrans à une vitesse constante, est également défini ainsi qu'une interface qui permet au skipper de notifier la consigne de cap `ExternalEvent` à suivre et de la modifier (à l'image des boîtiers de commande disponibles dans le commerce). L'environnement virtuel offre la possibilité à l'utilisateur d'agir à sa guise sur le vent, l'état de la mer et sur les réglages du voilier.

La définition des capteurs, de l'actionneur et des évènements externes, liés aux actions effectuées sur l'interface pilote, de la couche `cible` est ici directe.

5.2.2 *Etape 2 : construction de la bibliothèque de lois de commande et de la table des paramètres*

Au niveau du contrôle, le seul actionneur à commander est le vérin. Mais, les nombreux capteurs autorisent de multiples régulations. L'expertise a permis d'opter pour de la régulation de type PID car il s'agit d'une solution industrielle éprouvée et ses avantages et inconvénients sont bien connus par les marins donc exploitables. Pour autant, il existe de nombreuses implémentations (PPIDD, PDD, ...) de cette loi. Par exemple, le mode *cap* est une régulation de type PDD qui tient compte du roulis à cause de son importance pour l'anticipation :

$$\theta_{barre} = K_c(K_{cap}(cap - consCap) + K_{lacet} lacet + am.K_{roulis} roulis)$$

θ_{barre} est l'angle de barre absolu, `cap` la `data` donnant le cap magnétique du bateau, `consCap` la consigne de cap à respecter, `lacet` la vitesse de rotation en degrés par seconde du bateau par rapport à un axe vertical (lié à la centrale inertielle) et `roulis` la vitesse de rotation par rapport à l'axe du bateau (dirigé de l'arrière vers l'avant). K_{cap}, K_{lacet}, K_{roulis} et le gain K_c sont ajustés en fonction de l'environnement par le barreur adaptatif. Le terme am devant K_{roulis} vaut ± 1 en fonction de l'amure du bateau. Pour déterminer les bonnes lois et les paramétrer, une table de mixage (cf. figure 9) est créée. C'est l'interface du composant de mise au point. En sélectionnant le bouton correspondant, le mode de fonctionnement courant est choisi (les modes sont ici appelés `Cap`, `Gîte`, `Vitesse`, `Vent`, `Gîte rel` et `Vitesse rel` sur la figure). Et les différents paramètres ou coefficients (indiqués ici avec à leur gauche les modes dans lesquels ils interviennent) sont réglables à l'aide de *scroll bar*. Les 6 modes de barre retenus sont détaillés dans la table 1.

Une partie de la table des paramètres choisis est donnée en aperçu dans la table 2.

5.2.3 *Etape 3 : caractérisation des données en fonction des capteurs*

De nombreuses données sont presque directement définissables à partir des capteurs au formatage près. C'est le cas de la vitesse, de la position, du cap, de la force et de la direction du vent apparent... D'autres données sont plus délicates à obtenir. Par exemple la direction et la force du vent réel sont issues d'un calcul faisant intervenir vitesse, cap du bateau et angle et vitesse du vent réel. Le résultat peut être

4. ARéVi (Atelier de Réalité Virtuelle) est une bibliothèque de simulation et de rendu 3D.

5. Les jauges de contraintes servent à mesurer les déformations des matériaux sur lesquels elles sont fixées.

MODES DE BARRE
Mode cap :
$\theta_{barre} = K_c(K_{cap}(cap - consCap) + K_{lacet}lacet + am.K_{roulis}roulis)$
Mode cap relatif :
$\theta_{barre} = K_c(K_{cap}(cap - consCap) + K_{lacet}lacet + am.K_{roulis}roulis) + \theta_{reel}$
Mode vent réel :
$\theta_{barre} = K_{vr}(K_{TWA}(TWA - consTWA) + K_{lacet}lacet + am.K_{roulis}roulis)$
Mode gîte :
$\theta_{barre} = sgn.K_g(K_{gite}(gite - consGite) + am.K_{roulis}roulis + K_{sb}(sensabarre - consSB) + K_{dsb}(\frac{dsensabarre}{dt}))$
Mode gîte relatif :
$\theta_{barre} = sgn.K_{gr}(K_{gite}(gite - consGite) + am.K_{roulis}roulis + K_{sb}(sensabarre - consSB) + K_{dsb}(\frac{dsensabarre}{dt})) + \theta_{reel}$
Mode gain :
$\theta_{barre} = -K_{gr}(K_{gite}(gite - consGite) + am.K_{roulis}roulis + K_{sb}(sensabarre - consSB) + K_{dsb}(\frac{dsensabarre}{dt})) + \theta_{reel}$

Table 1: La bibliothèque du barreur réactif

	Près Mer Calme	Largue Mer Calme	...	Près Mer Agitée	default
Mode cap	$K_c = 1$ $K_{cap} = 5$ $K_{lacet} = 10$ $K_{roulis} = 0$	$K_c = 1$ $K_{cap} = 5$ $K_{lacet} = 15$ $K_{roulis} = 10$	...	$K_c = 2$ $K_{cap} = 5$ $K_{lacet} = 10$ $K_{roulis} = 5$	$K_c = 1.25$ $K_{cap} = 5$ $K_{lacet} = 10$ $K_{roulis} = 0$
Mode vent réel	$K_{vr} = 1$ $K_{TWA} = 5$ $K_{lacet} = 20$ $K_{roulis} = 0$	$K_{vr} = 1.2$ $K_{TWA} = 5$ $K_{lacet} = 20$ $K_{roulis} = 20$	...	$K_{vr} = 2$ $K_{TWA} = 5$ $K_{lacet} = 10$ $K_{roulis} = 0$	$K_{vr} = 1.25$ $K_{TWA} = 5$ $K_{lacet} = 20$ $K_{roulis} = 0$

Table 2: Extrait de la table des paramètres

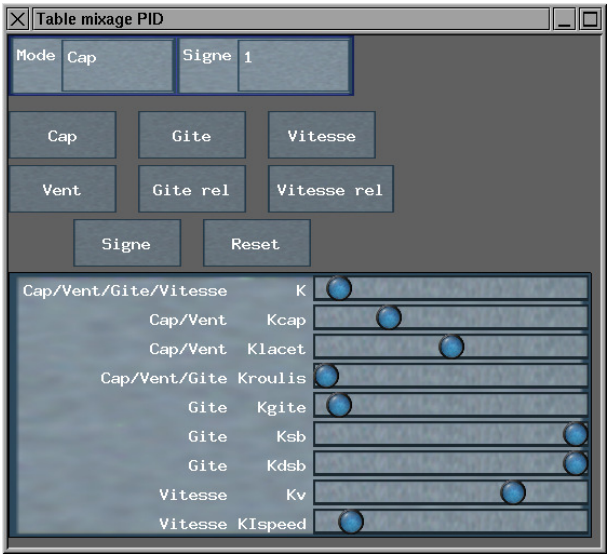


Figure 9: Table de mixage employée pour le réglage des PID

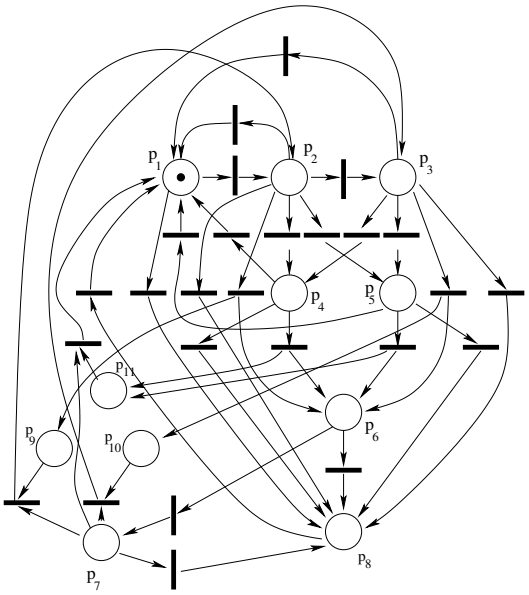


Figure 10: Automate contrôlant le barreur expert

affiné en exploitant la centrale inertielle pour tenir compte de l'attitude du bateau.

De plus,un certain nombre de contextes tel que `Largue et Mer Calme` sont nécessaires à l'adaptation des modes. Un contexte combine plusieurs informations. Pour `Largue et Mer Calme`, la première concerne les *allures* et la deuxième les *états de mer*. Chaque contexte correspond au final à une `donnée` de type énuméré. Certaines comme l'*état de mer* sont basées sur une analyse fréquentielle des mouvements du bateau et sur une double intégration sur des accélérations fournies par la centrale inertielle pour estimer la hauteur des vagues. Ensuite l'échelle de Douglas [6] sert de référence pour qualifier les différents états de mer. D'autres comme les *allures* du bateau sont basées sur l'angle au vent réel moyenné sur une dizaine de secondes.

Les données nécessaires sont au final très nombreuses. Outre les données continues qui donnent des valeurs de type capteurs, celles de type énuméré peuvent se classer en 5 grandes catégories en fonction de ce qu'elles permettent de caractériser : le bateau, la mer, le vent, la route du bateau et les contextes pour les modes de barre. Cette classification est relative au métier et non intégrée à l'architecture. Elle apparait dans les librairies de modèle et de code spécifique.

5.2.4 *Etape 4 : construction de l'automate et définition des événements*

La dernière étape porte sur la construction de l'automate, c'est-à-dire le `barreur expert`. La figure 10 en donne une version simplifiée sous la forme d'un réseau de Petri interprété. Dans cette vue, les transitions ne sont pas étiquetées.

Cette étape oblige à définir les derniers événements nécessaires et donc quelques nouvelles données. Par exemple le passage entre le mode *cap* et le mode *vent réel* peut se faire sur une notion de stabilité liée à la fois au vent, à la mer, aux mouvements de la barre et à l'allure du bateau. Les consignes se calculent elles directement à partir des données.

Cette mise au point incrémentale justifie pleinement l'utilisation de l'architectural, de la librairie métier et de l'environnement de mise au point. Le développement devient alors un assemblage formaté et adaptable de composants logiciels métier.

5.3 Expérimentation

L'environnement virtuel permet de tester le pilote dans des conditions de mer et de vent variées. Différents éléments du bateau sont réglables comme les voiles, la quille ou la position du skipper. Cependant, tester la qualité d'un pilotage est difficile car il s'agit toujours d'un compromis entre plusieurs critères comme performance, sécurité, consommation énergétique et trajectoire. Aussi le bateau est cloné pour pouvoir comparer le pilote construit selon l'architecture IMOCA, appelé *barreur virtuel*, avec un simple régulateur de cap proche d'un pilote commercial, appelé *PID*. Un certain nombre de cas (rafale, départ en surf, clapot ...) connus pour prendre en défaut ces pilotes commerciaux servent pour la comparaison. La figure 11 présente la trajectoire du clone ainsi que du *barreur virtuel*. Les deux voiliers naviguent travers au vent dans 20 noeuds de vent par mer calme lorsque survient une rafale à 23 noeuds qui dure une quarantaine de secondes avant que le vent ne reprenne sa force initiale. Au départ (à gauche sur la figure) les deux voiliers sont quasiment superposés. Quand la rafale arrive ils abattent légèrement pour essayer de contrecarrer le surcroît de puissance mais ils finissent par partir au lof, c'est-à-dire par se tourner brutalement vers le vent. Le *barreur virtuel* passe en mode gîte et accompagne cette aulofée alors que le *PID* cherche à lutter contre l'écart à la consigne en donnant beaucoup de barre (voir la figure 12). Lorsque la rafale se termine le *barreur virtuel* repasse en mode cap et le clone finit par venir s'aligner à l'arrière de l'autre voilier.

6. L'échelle de Douglas est celle mondialement utilisée dans les bulletins météorologiques.

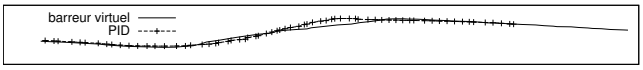


Figure 11: Trajectoires des deux voiliers lors d'une rafale

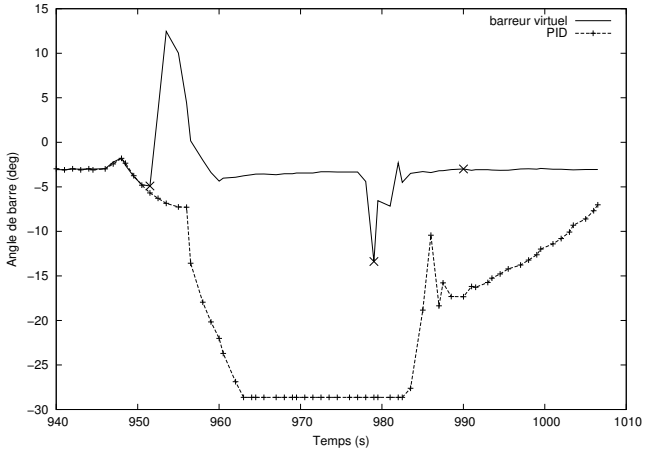


Figure 12: Angles de barre

Les courbes des figures 12 et 13 donnent une explication de ce qui se passe en donnant les angles de barre et les vitesses des deux bateaux.

Alors que le *barreur virtuel* change de mode (les changements de mode apparaissent avec des croix en X) et parvient ainsi à conserver une bonne vitesse, le *PID* cherche à compenser la déviation de trajectoire en donnant beaucoup de barre ce qui freine le bateau.

6. CONCLUSION

Ce document présente l'architecture IMOCA et une méthodologie dédiés aux systèmes de contrôle de processus par régulation en environnement perturbé. Le principe est d'effectuer de la commutation de modes en fonction des évolutions de l'environnement ; les modes, leurs paramètres, leur enchaînement étant déterminés par expertise du domaine d'application.

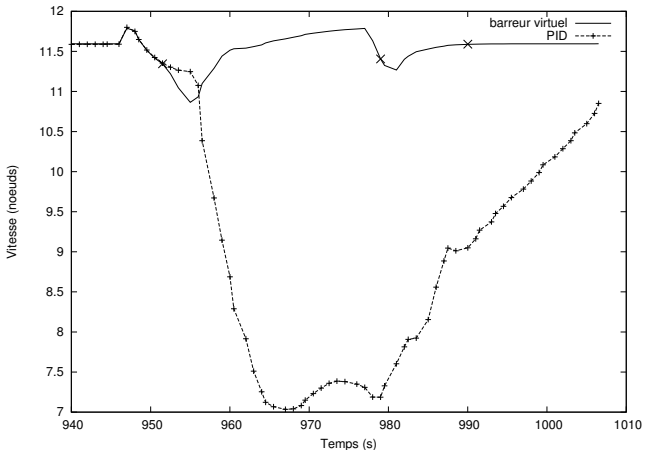


Figure 13: Vitesses

IMOCA offre des modèles et des outils pour la mise au point incrémentale des données et des paramètres du système de contrôle. Cette architecture et cette méthodologie ont été appliquées au cas du développement d'un pilote automatique pour voiliers et donnent un résultat opérationnel apportant un gain à la fois en performance mais également en sécurité.

Dans les perspectives, nous visons la mise en place d'un code opérationnel embarquable, adaptable et reconfigurable sur voilier réel. Nous pensons aussi augmenter les librairies disponibles pour viser d'autres domaines d'application.

7. REFERENCES

[1] D. Bertrand, A.-M. Déplanche, S. Faucou, and O. H. Roux. A study of the AADL mode change protocol. In *Proceedings of the IEEE International Conference on Engineering Complex Computer Systems - ICECCS 2008*, page 288, Belfast, Ireland, 2008. IEEE Computer Society.

[2] J. Bézivin. In Search of a Basic Principle for Model Driven Engineering. *The European Journal for the Informatics Professional*, 2 :21–24, 2004.

[3] E. Borde, G. Haik, and L. Pautet. Mode-based reconfiguration of critical software component architectures. In *Design, Automation Test in Europe Conference Exhibition, 2009. DATE '09.*, pages 1160–1165, 2009.

[4] Y. Brun, G. Marzo Serugendo, C. Gacek, H. Giese, H. Kienle, M. Litoiu, H. Müller, M. Pezzè, and M. Shaw. Software Engineering for Self-Adaptive Systems. chapter Engineering Self-Adaptive Systems through Feedback Loops, pages 48–70. 2009.

[5] R. A. D. Garlan and J. Ockerbloom. Exploiting style in architectural design environments. In *Proceedings of the 2nd ACM SIGSOFT symposium on Foundations of software engineering*, page 175–188, New York, NY, USA, 1994.

[6] R. David and H. Alla. *Du Grafcet aux Réseaux de Petri*. Hermès, 1992.

[7] J. Deantoni and J.-P. Babau. A MDA-based approach for real time embedded systems simulation. In *Proceedings of the 9th IEEE International Symposium on Distributed Simulation and Real-Time Applications*, pages 257–264, Montreal, 2005. IEEE Computer Society.

[8] J. Deantoni and J.-P. Babau. SAIA : safe deployment of sensors based real time application. In *Workshop on Models and Analysis for Automotive Systems (held in conjunction with RTSS)*, Rio de Janeiro, Brésil, Dec. 2006.

[9] G. H. E. Borde and L. Pautet. Mode-Based Reconfiguration of Critical Sofware Component Architectures. In *Design, Automation Test in Europe Conference Exhibition*, pages 20–24, 2009.

[10] V. A. F Maker, R. Amirtharajah. MELOADES : Methodology for Long-term Online Adaptation of Embedded Software for Heterogeneous Devices. Technical Report CSE-2013-69, UC Davis, Computer Science, 2012.

[11] G. Guillou. *Architecture multi-agents pour le pilotage*

automatique des voiliers de compétition et Extensions algébriques des réseaux de Petri. PhD thesis, Université de Bretagne Occidentale, 2010.

[12] L. Holloway, B. Krogh, and A. Giua. A Survey of Petri Net Methods for Controlled Discrete Event Systems. *Discrete Event Dynamic Systems*, 7(2) :151–190, 1997.

[13] P. Horn. Autonomic Computing : IBM's Perspective on the State of Information Technology, 2001.

[14] M. v. S. J.P. Almeida, R. Dijkman and L. Pires. On the notion of abstract platform in MDA development. In *Proceedings of the eighth IEEE International Enterprise Distributed Object Computing Conference*, pages 253–263, 2004.

[15] Y. Li, K. H. Ang, and G. C. Y. Chong. PID control system analysis and design. *IEEE Control Systems Magazine*, 26(1) :32–41, 2006.

[16] L. M.Agrawal, S.Cooper and V. Thomas. An open software architecture for high-integrity and high-availability avionics. In *Proceedings of DASC04, Digital avionics Systems Conference*, 2004.

[17] OMG. Model driven architecture guide v1.0.1, 2003.

[18] S. V. P. H. Feiler, B. Lewis and E. Colbert. An overview of the SAE Architecture & Design Language (AADL) Standard : A Basis for Model Based Architecture-Driven Embedded Systems Engineering. In *Architecture Description Language, workshop at IFIP World Computer Congress*, 2004.

[19] M. Parenthoën. *Animation phénomènologique de la mer : une approche énactive*. PhD thesis, Université de Bretagne Occidentale, 2004.

[20] F. Terrier and S. Gérard. MDE benefits for didtributed, real time and embedded systems, 2006.